Date=17/07/2020
Lecture By=Shubham Joshi
Subject ⇒Time Complexity

IN PREVIOUS LECTURE **(QUICK RECAP)** **Date-14/07/2020**	**In Today's Lecture (Overview)**
Revision Of Previous lectures For Python	⇒ SLA in Software ⇒ What Is Time Complexity In Python? ⇒ What Big O Notation Means? ==>What Are The Different Big O Notation Measures?' O(1): O(log n): O(n): (n log n): O(n square): ⇒ Examples

---

⇒ SLA in Software

-A **software service level agreement** (**SLA**) is a contract between your business and your IT supplier.
-A service-level agreement (SLA) defines the level of service you expect from a vendor
-"Click Here" To know more about It

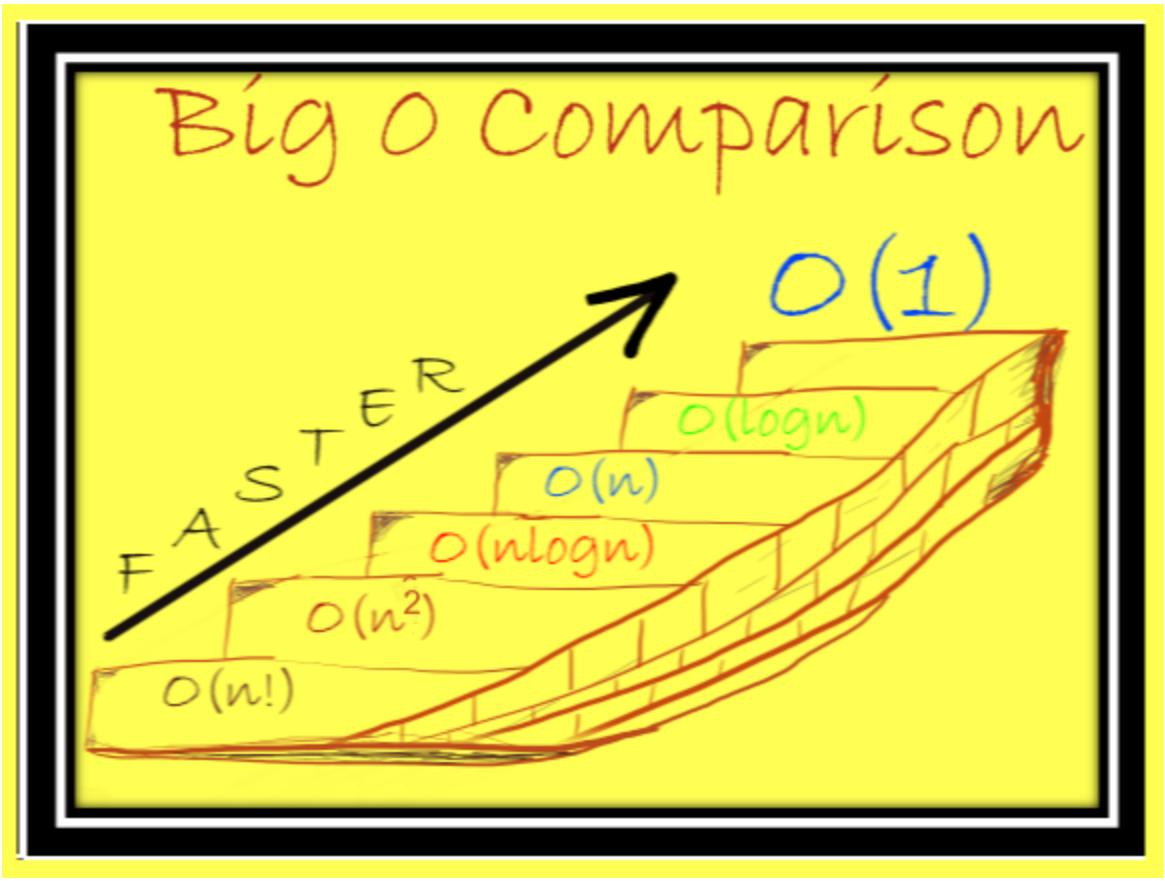
⇒ What Is Time Complexity In Python?

-Time complexity is commonly estimated by c**ounting the number of elementary operations performed by the algorithm**
-supposing that each elementary operation takes a fixed amount of time to perform.

⇒ What Big O Notation Means?

-A number of operations are performed in an algorithm.
-Big-O measures the time complexity of the operations of an algorithm.
-it measures how long it takes for the algorithm to compute the required operation.

==> **In simplest terms, Big O notation is a way to measure performance of an operation based on the input size, known as n.**



Big - O Notation	Computations for 10 Elements	Computations For 100 Elements	Computations For 1000 Elements
O(1)	1	1	1
O(N)	10	100	1000
O(N^2)	100	10000	1000000
O(log N)	3	6	9
O(N log N)	30	600	9000
O(2^N)	1024	1.26e+29	1.07e+301
O(N!)	3628800	9.33e+157	4.02e+2567

==>What Are The Different Big O Notation Measures?'

Name	Time Complexity
Constant Time	O(1)
Logarithmic Time	O(log n)
Linear Time	O(n)
Quasilinear Time	O(n log n)
Quadratic Time	O(n^2)

-**consider n to be the size of the input** collection. In terms of time complexity:

O(1):

No matter how big your collection is, the time it takes to perform an operation is constant.
- As an instance, operations that check whether a collection has any items inside it is an O(1) operation.

Example

```
if a > b:
   return True
else:
   return False
```

O(log n):

When the size of a collection increases, the time it takes to perform an operation increases logarithmically.
-This is the logarithmic time complexity notation. Potentially optimised searching algorithms are O(log n).

Example

```
for index in range(0, len(data), 3):
   print(data[index])
```

O(n):

The time it takes to perform an operation is directly and linearly proportional to the number of items in the collection.
-if we want to sum all of the items in a collection then we would have to iterate over the collection. Hence the iteration of a collection is an O(n) operation.

Example

```
for value in data:
   print(value)
```

(n log n):

Where the performance of performing an operation is a quasilinear function of the number of items in the collection.
- Time complexity of an optimised sorting algorithm is usually n(log n).

Example

```
for value in data1:
   result.append(binary_search(data2, value))
```

O(n square):

When the time it takes to perform an operation is proportional to the square of the items in the collection. This is known as the quadratic time complexity notation

Example

```
for x in data:
    for y in data:
        print(x, y)
```

“Click Here” To Know More about it

“Click Here” for video Tutorial

⇒ Examples

1 .**a = 0 , i = N**

```
while i > 0 :
  a += i
   i /= 2
```

Complexity Of This Code is logn

2.**i = n / 2**

```
while i < n:
   j = 2
   while j < n:
     j *= 2
```

Complexity Of this Code is Nlogn

3.**count = 0 ;**

```
i = N
while i > 0:
  While j < i:
 count+=1
       i /= 2
```

== Complexity Of this Code is O(N)